

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики  
Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №3**  
з дисципліни “Бази даних”  
тема “Засоби оптимізації роботи СУБД PostgreSQL”

Виконав(ла)  
студент(ка) II курсу  
групи КП-91

Чорна Софія Олександрівна

*(прізвище, ім'я, по батькові)*

варіант №24

Перевірів  
“ \_\_\_\_\_ ” “ \_\_\_\_\_ ” 20\_\_ р.  
викладач

Петрашенко Андрій Васильович

*(прізвище, ім'я, по батькові)*

Київ 2020

*Завдання роботи полягає у наступному:*

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

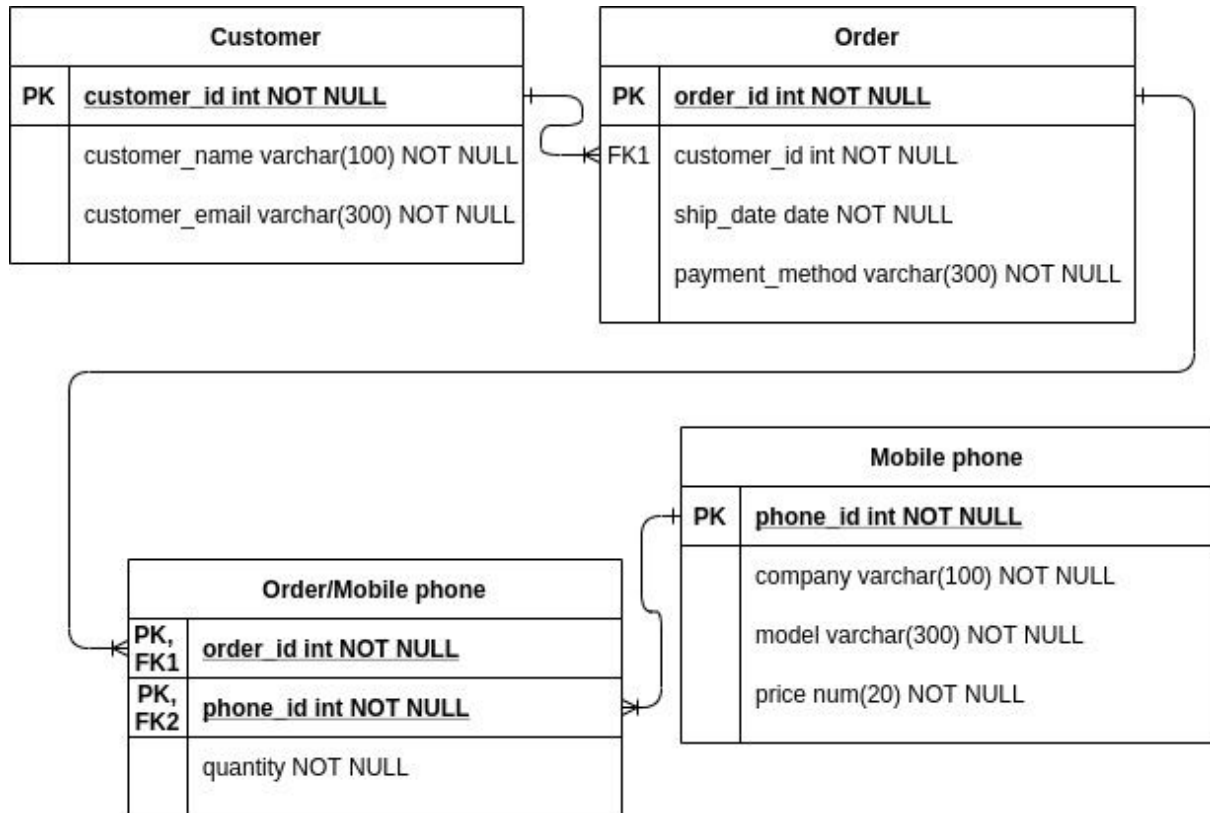
Варіант №24

Посилання на репозиторій:

<https://github.com/schernaya/databases/tree/master/lab3>

Пункт 1:

Схема бази даних:



ORM класи:

```
class Customer(Base):
    __tablename__ = 'customers'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)

    forms = relationship("Form")

    def __init__(self, id, name, email):
        self.id = id
        self.name = name
        self.email = email

    def __repr__(self):
        return "<Customer(id={}, name='{}', email='{}')>" \
            .format(self.id, self.name, self.email)
```

```

class Form(Base):
    __tablename__ = 'forms'
    id = Column(Integer, primary_key=True)
    payment_method = Column(String)
    ship_date = Column(Date)
    customer_id = Column(Integer, ForeignKey('customers.id'))

    phones = relationship('Phone', secondary=form_phone_links, back_populates='forms')

    def __init__(self, id, payment_method, ship_date, customer_id):
        self.id = id
        self.payment_method = payment_method
        self.ship_date = ship_date
        self.customer_id = customer_id

    def __repr__(self):
        return "<Form(id={}, payment_method='{ }', ship_date='{ }', customer_id={})>" \
            .format(self.id, self.payment_method, self.ship_date, self.customer_id)

```

```

class Phone(Base):
    __tablename__ = 'phones'
    id = Column(Integer, primary_key=True)
    model = Column(String)
    company = Column(String)
    price = Column(Integer)

    forms = relationship("Form", secondary=form_phone_links, back_populates="phones")

    def __init__(self, id, model, company, price):
        self.id = id
        self.model = model
        self.company = company
        self.price = price

    def __repr__(self):
        return "<Phone(id={}, model='{ }', company='{ }', price={})>" \
            .format(self.id, self.model, self.company, self.price)

```

```

form_phone_links = Table('form_phone_links', Base.metadata,
    Column('form_id', Integer, ForeignKey('forms.id')),
    Column('phone_id', Integer, ForeignKey('phones.id')),
    Column('quantity', Integer)
)

```

Приклади ORM запитів:

```
def read_customers(self):
    return self.session.query(Customer).all()

def read_forms_phone(self, form_id):
    form = self.session.query(Form).get(form_id)
    if form is None:
        return []
    return form.phones

def create_phone(self, item):
    self.session.add(item)
    self.session.commit()
    return item

def create_form_phone(self, form_id, phone_id, quantity):
    req = form_phone_links.insert() \
        .returning(form_phone_links.c.form_id,
                    form_phone_links.c.phone_id,
                    form_phone_links.c.quantity).values(
        form_id=form_id, phone_id=phone_id, quantity=quantity)
    res = self.session.execute(req)
    self.session.commit()
    return res

def update_form(self, item):
    self.session.query(Form) \
        .filter(Form.id == item.id) \
        .update({
            Form.payment_method: item.payment_method,
            Form.ship_date: item.ship_date,
            Form.customer_id: item.customer_id
        })
    form = self.session.query(Form).get(item.id)
    return form
```

```
def delete_form(self, item_id):
    self.session.query(Form).filter(Form.id == item_id).delete()
    res = self.session.commit()
    return res
```

Пункт 2:

GIN

Команда створення:

```
CREATE INDEX textsearch_idx ON customers USING GIN (ts_index_col)
```





Було створено окремий стовпець `ts_index_col`, в якому зберігається результат `to_tsvector`, а також створено тригер, який буде підтримувати стовпець з `tsvector` в актуальному стані при будь-яких змінах рядка.

```
CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE
ON customers FOR EACH ROW EXECUTE PROCEDURE
tsvector_update_trigger(ts_index_col, 'pg_catalog.english', name, email);
```

Приклади запитів:

1)

```
1 SELECT * FROM customers
2 WHERE ts_index_col @@ to_tsquery('Sophie')
3 ORDER BY customers.id DESC
4
```

Data Output		Explain	Messages	Notifications				
	id [PK] integer		name character varying (100)		email character varying (300)		ts_index_col tsvector	
1		100066	Sophie Chorna		user_99931@gmail.com		'chorna':2 'sophi':1	
2		100040	Sophie Chorna		user_99905@yahoo.com		'chorna':2 'sophi':1	
3		100032	Sophie Chorna		user_99897@hotmail.com		'chorna':2 'sophi':1	
4		100028	Sophie Chorna		user_99893@hotmail.com		'chorna':2 'sophi':1	
5		100017	Sophie Chorna		user_99882@gmail.com		'chorna':2 'sophi':1	
6		100015	Sophie Chorna		user_99880@hotmail.com		'chorna':2 'sophi':1	
7		100005	Sophie Chorna		user_99870@ex.ua		'chorna':2 'sophi':1	



```

1 EXPLAIN ANALYZE
2 SELECT * FROM customers
3 WHERE ts_index_col @@ to_tsquery('Sophie')
4 ORDER BY customers.id DESC
5

```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Sort (cost=5605.28..5629.76 rows=9791 width=78) (actual time=7.155..7.976 rows=9924 loops=1)
2	Sort Key: id DESC
3	Sort Method: quicksort Memory: 1780kB
4	-> Bitmap Heap Scan on customers (cost=100.13..4956.27 rows=9791 width=78) (actual time=1.144..4.482 rows=9924 loops=1)
5	Recheck Cond: (ts_index_col @@ to_tsquery('Sophie':text))
6	Heap Blocks: exact=1375
7	-> Bitmap Index Scan on textsearch_idx (cost=0.00..97.69 rows=9791 width=0) (actual time=0.962..0.963 rows=9924 loops=1)
8	Index Cond: (ts_index_col @@ to_tsquery('Sophie':text))
9	Planning time: 0.101 ms
10	Execution time: 8.391 ms

Bez GIN, execution time: 12.035 ms

2)

```

1 select * from customers
2 where ts_index_col @@ plainto_tsquery('petrovich')
3 order by ts_rank(ts_index_col, plainto_tsquery('petrovich')) DESC
4

```

Data Output Explain Messages Notifications

	id	name	email	ts_index_col
	[PK] integer	character varying (100)	character varying (300)	tsvector
1	35	Viktor Petrovich	user_1@hotmail.com	'petrovich':2 'viktor':1
2	43	Viktor Petrovich	user_9@hotmail.com	'petrovich':2 'viktor':1
3	45	Viktor Petrovich	user_11@yahoo.com	'petrovich':2 'viktor':1
4	46	Viktor Petrovich	user_12@ex.ua	'petrovich':2 'viktor':1

```

1 explain analyze
2 select * from customers
3 where ts_index_col @@ plainto_tsquery('petrovich')
4 order by ts_rank(ts_index_col, plainto_tsquery('petrovich')) DESC

```

Data Output Explain Messages Notifications

QUERY PLAN	
	text
1	Gather Merge (cost=6821.24..7504.57 rows=5942 width=82) (actual time=16.864..21.913 rows=10023 loops=1)
2	Workers Planned: 1
3	Workers Launched: 1
4	-> Sort (cost=5821.23..5836.08 rows=5942 width=82) (actual time=14.437..14.786 rows=5012 loops=2)
5	Sort Key: (ts_rank(ts_index_col, plainto_tsquery('petrovich':text))) DESC
6	Sort Method: quicksort Memory: 972kB
7	-> Parallel Bitmap Heap Scan on customers (cost=102.54..5448.76 rows=5942 width=82) (actual time=1.471..12.612 rows=5012 loops=2)
8	Recheck Cond: (ts_index_col @@ plainto_tsquery('petrovich':text))
9	Heap Blocks: exact=763
10	-> Bitmap Index Scan on textsearch_idx (cost=0.00..100.01 rows=10102 width=0) (actual time=2.218..2.219 rows=10027 loops=1)
11	Index Cond: (ts_index_col @@ plainto_tsquery('petrovich':text))
12	Planning time: 0.157 ms
13	Execution time: 22.545 ms

Без GIN, execution time: 30.039 ms

Отже, індекс GIN прискорює виконання запиту. Мало ряд переваг зберігання обчисленого виразу індексу в окремому стовпці. По-перше, для використання індексу в запитах не потрібно явно вказувати ім'я конфігурації текстового пошуку. По-друге, пошук виконується швидше, так як для перевірки відповідності даних індексу не потрібно повторно виконувати `to_tsvector`.

## BRIN

Перед створенням треба перевірити, для якого стовпця доречно створювати BRIN індекс. З таблиць та їхніх стовпців `customers.email` має значення кореляції біля одиниці.



```

1 select attname, correlation from pg_stats
2 where tablename='customers'
3 order by correlation desc nulls last
4

```

Data Output Explain Messages Notifications

	attname name	correlation real
1	id	0.998992
2	email	0.817564
3	name	0.162785

Команда створення:

```

create index brin_indx on customers using brin(email)

```

Приклади запитів:

1)

```

1 select * from customers where email LIKE '%user_%'
2 order by customers.id ASC
3

```

Data Output Explain Messages Notifications

	id [PK] integer	name character varying (100)	email character varying (300)	ts_index_col tsvector
1	35	Viktor Petrovich	user_1@hotmail.com	'petrovich':2 'viktor':1
2	36	Tatiana Nikolaevna	user_2@ex.ua	'nikolaevna':2 'tatiana':1
3	37	Ruslan Anatolich	user_3@yahoo.com	'anatolich':2 'ruslan':1
4	38	Tatiana Nikolaevna	user_4@hotmail.com	'nikolaevna':2 'tatiana':1
5	39	Sophie	user_5@hotmail.com	'sophi':1
6	40	Andrii Vasylovych	user_6@hotmail.com	'andrii':1 'vasylovych':2
7	41	Nataliya Antonivna	user_7@yahoo.com	'antonivna':2 'nataliya':1
8	42	Ruslan Anatolich	user_8@hotmail.com	'anatolich':2 'ruslan':1
9	43	Viktor Petrovich	user_9@hotmail.com	'petrovich':2 'viktor':1

```

1 explain analyze select * from customers where email LIKE '%user_%'
2 order by customers.id ASC
3

```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	Index Scan using "Customer_pkey" on customers (cost=0.42..7354.67 rows=100072 width=78) (actual time=0.015..24.026 rows=100076 loo...	
2	Filter: ((email)::text ~~ '%user_%':text)	
3	Rows Removed by Filter: 6	
4	Planning time: 0.095 ms	
5	Execution time: 27.124 ms	

2)

```

1 select email, count(*) from customers
2 where email like '%gmail.com%' group by email
3

```

Data Output Explain Messages Notifications

	email	count	
	character varying (300)	bigint	
1	user_28704@gmail.com	1	
2	user_78005@gmail.com	1	
3	user_5749@gmail.com	1	
4	user_78785@gmail.com	1	
5	user_70746@gmail.com	1	
6	user_49809@gmail.com	1	
7	user_10215@gmail.com	1	

```

1 explain analyze
2 select email, count(*) from customers
3 where email like '%gmail.com%' group by email
4

```

Data Output Explain Messages Notifications

	QUERY PLAN	
	text	
1	HashAggregate (cost=3617.89..3779.61 rows=16172 width=28) (actual time=19.724..22.223 rows=16684 loops=1)	
2	Group Key: email	
3	-> Seq Scan on customers (cost=0.00..3537.03 rows=16173 width=20) (actual time=0.011..14.527 rows=16685 loops=1)	
4	Filter: ((email)::text ~~ '%gmail.com%':text)	
5	Rows Removed by Filter: 83397	
6	Planning time: 0.086 ms	
7	Execution time: 22.815 ms	

Отже, індекс BRIN прискорює виконання запиту, адже було обрано стовпець, для якого використовувати його раціонально. BRIN створює «сторінки» значень, де помічає мінімальне та максимальне значення на сторінці, що пришвидшує пошук відсортованих даних.

*Пункт 3:*

Створення тригера:

```
CREATE OR REPLACE FUNCTION tfunc_usr() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.name IS NULL THEN
        RAISE EXCEPTION 'name cannot be null';
    END IF;

    IF NOT (NEW.email LIKE '%@%') THEN
        RAISE EXCEPTION '% incorrect email', NEW.email;
    END IF;

    IF (TG_OP = 'UPDATE') THEN
        INSERT INTO usr_audit VALUES('U', user, NEW.name, NEW.email, now());
        RETURN NEW;

    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO usr_audit VALUES('I', user, NEW.name, NEW.email, now());
        RETURN NEW;

    END IF;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER user_trigger
AFTER INSERT OR UPDATE ON customers|
FOR EACH ROW EXECUTE PROCEDURE tfunc_usr();
```

Створений тригер гарантує, що будь-яке додавання або зміна рядка в таблиці customers буде зафіксовано в таблиці usr\_audit. Також він фіксує поточний час, ім'я користувача і тип виконуваної операції.

Приклади запитів, що ініціюють тригер та результати його виконання:

1) Перед запитом для додавання:

```
1 SELECT * FROM public.usr_audit
2
```

Data Output Explain Messages Notifications

	operation character (1)	userid text	usname character varying	email character varying	stamp timestamp without time zone
1	I	postgres	name	ew@re	2020-11-26 01:13:15.861314
2	I	postgres	rw	er@e	2020-11-26 01:13:50.157205
3	I	postgres	Alina	alya@gmail.com	2020-11-26 10:08:01.763566
4	U	postgres	Alina	imalina@gmail.com	2020-11-26 10:09:07.377449
5	U	postgres	Sophia	imalina@gmail.com	2020-11-26 10:10:38.25535

Запит:

```
INSERT INTO public.customers(name, email)
VALUES ('new name', 'someemail@gmail.com');
```

Після запиту:

```
1 SELECT * FROM public.usr_audit
2
```

Data Output Explain Messages Notifications

	operation character (1)	userid text	usname character varying	email character varying	stamp timestamp without time zone
1	I	postgres	name	ew@re	2020-11-26 01:13:15.861314
2	I	postgres	rw	er@e	2020-11-26 01:13:50.157205
3	I	postgres	Alina	alya@gmail.com	2020-11-26 10:08:01.763566
4	U	postgres	Alina	imalina@gmail.com	2020-11-26 10:09:07.377449
5	U	postgres	Sophia	imalina@gmail.com	2020-11-26 10:10:38.25535
6	I	postgres	new name	someemail@gmail.com	2020-11-26 11:30:58.844355

2) Запит:

```
UPDATE public.customers
SET email='mynewemail@gmail.com'
WHERE id = 153;
```

Після запиту:



```
1 SELECT * FROM public.usr_audit
2
```

Data Output Explain Messages Notifications

	operation character (1)	userid text	username character varying	email character varying	stamp timestamp without time zone
1	I	postgres	name	ew@re	2020-11-26 01:13:15.861314
2	I	postgres	rw	er@e	2020-11-26 01:13:50.157205
3	I	postgres	Alina	alya@gmail.com	2020-11-26 10:08:01.763566
4	U	postgres	Alina	imalina@gmail.com	2020-11-26 10:09:07.377449
5	U	postgres	Sophia	imalina@gmail.com	2020-11-26 10:10:38.25535
6	I	postgres	new name	someemail@gmail.com	2020-11-26 11:30:58.844355
7	U	postgres	Sophie Chorna	mynewemail@gmail.com	2020-11-26 11:35:00.017505

Приклад помилки:

```
1 UPDATE public.customers
2 SET email='wrongemail'
3 WHERE id = 153;
```

Data Output Explain Messages Notifications

```
ERROR: wrongemail incorrect email
CONTEXT: PL/pgSQL function tfunc_usr() line 8 at RAISE
SQL state: P0001
```

*Контрольні запитання*

1. Сформулювати призначення та задачі об'єктно-реляційної проекції (ORM).
2. Проаналізувати основні види індексів у PostgreSQL (*BTree*, *BRIN*, *GIN*, *Hash*): призначення, сфера застосування, переваги та недоліки.
3. Пояснити призначення тригерів та функцій у базах даних.

*Відповіді:*

1. ORM використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому ORM сама піклується про перетворення даних між двома несумісними станами. Більшість ORM-інструментів покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру

бази даних, а бази даних - нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому і база даних, і додаток можуть працювати з даними, кожен у своїй вихідній формі. Використання ORM в проекті позбавляє від необхідності роботи з SQL.

2.

## B-tree

Призначення: прискорення роботи з даними, які можна відсортувати

Сфера застосування: використовується у випадках, якщо до індексованих полів застосовуються: оператори порівняння, умови порожнечі, оператори пошуку підрядка

Недолік: займає деяку кількість пам'яті

## GIN

Призначення: для повнотекстового пошуку

Сфера застосування: текстові дані, серед яких відносна невелика кількість унікальних лексем; застосовується до складових типів, робота з якими здійснюється за допомогою ключів (масиви, jsonb і tsvector)

Недолік: повільне оновлення індексу при додаванні нового документу

## Hash

Призначення: створення хеш-таблиць для даних для пришвидшення доступу до даних за ключем

Сфера застосування: використовується тільки якщо проіндексоване поле бере участь в порівнянні (тільки оператор =).

Недолік: великий об'ємом займає пам'яті, не використовується в умовах IS NULL і IS NOT NULL



Brin

Призначення: прискорення пошуку строк, шляхом відкидання завідомо не підходящі строки

Сфера застосування: призначений для обробки дуже великих таблиць, у яких певні стовпці мають певну природну кореляцію з їх фізичним розташуванням у таблиці

Недолік: працює повільніше ніж В-дерево, але й займає менше місця. Через цю особливість для невеликих таблиць, використання В-дерева буде доцільним.

3. Триггер - це особлива різнобічність збережених процедур у базі даних. Тригери викликаються БД при певних обставинах для обробки виключних ситуацій, виконання необхідних дій після\до дії, для запису та створення нових таблиць для подальшого аналізу даних та надання додаткової інформації від БД. До команд маніпуляцій даними відносяться: UPDATE, INSERT, DELETE і SELECT.